# Introduction to security and cryptography
# Lecture 3: Hash functions, Message Authenticated Codes

Marius Lombard-Platet

ENS, Almerys

# Hash functions

# Hash Functions

**Definition**
A hash function $H$ takes as input a bit-string of any finite length and returns a corresponding *digest* of **fixed length**.

$$h : \{0,1\}^* \rightarrow \{0,1\}^n$$

# Hash Functions

**Definition**
A hash function $H$ takes as input a bit-string of any finite length and returns a corresponding *digest* of **fixed length**.

$$h : \{0,1\}^* \to \{0,1\}^n$$

**Example**

$$h \colon \{0,1\}^* \to \{0,1\}$$
$$b_1 \dots b_n \mapsto b_1$$

# Properties of Cryptographic Hash Functions (1/3)

**Pre-image resistance**
Given an output $y$, it is computationally infeasible to compute $x$ such that

$$h(x) = y$$

# Properties of Cryptographic Hash Functions (2/3)

**Second Pre-image resistance**
Given an input $x$, it is computationally infeasible to compute $x'$ such that

$$h(x') = h(x)$$

# Properties of Cryptographic Hash Functions (3/3)

**Collision resistance**
It is computationally infeasible to compute $x$ and $x'$ such that

$$h(x) = h(x')$$

# Properties of Cryptographic Hash Functions (consequence)

**Avalanche effect**
If we flip one bit of a message $x$ to get $x'$, the hashes $h(x)$ and $h(x')$ will be completely different.

Ex:

- SHA3-256(1) = 67b176705b46206614219f47a05aee7ae6a3edbe850bbbe214c536b989aea4d2
- SHA3-256(2) = b1b1bd1ed240b1496c81ccf19ceccf2af6fd24fac10ae42023628abbe2687310

# Note on hash functions

Hash functions are not used for encryption! They are used for *fingerprinting*: from an arbitrary long message or document, we always obtain a digest of small size (a few bits).

Goal of a hash function: two different messages will have different hashes, and from the hash one cannot retreive the original message.

# Merkle-Damgård Construction (variant for SHA, MD5)

Let $f$ be a compression fonction

$$f : \{0,1\}^{n+k} \rightarrow \{0,1\}^n$$

# Merkle-Damgård Construction (variant for SHA, MD5)

Let $f$ be a compression fonction

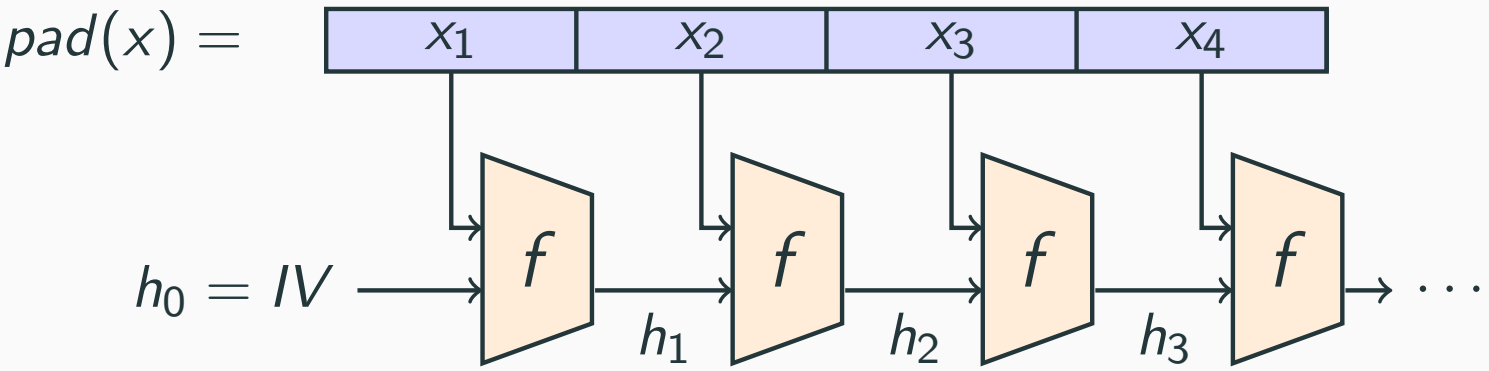
$$f : \{0,1\}^{n+k} \rightarrow \{0,1\}^n$$

1. Break the message $x$ to hash in blocks of size $k$:
   $x = x_1 x_2 \dots x_t$
2. Pad $x_t$ with a specific padding[1]
3. Define $x_{t+1}$ as the binary representation of the bit length of $x$
4. Iterate over the blocks

$$\begin{aligned} h_0 &= 0^n \text{ or any IV} \\ h_i &= f(h_{i-1} || x_i) \\ h(x) &= h_{t+1} \end{aligned}$$

[1] Notably, the final size of the last block must be $k$

# Merkle-Damgård Construction



# Merkle-Damgård Construction

**Examples**

- MD5 (1991)
- SHA-1 (1995)

# Merkle-Damgård Construction

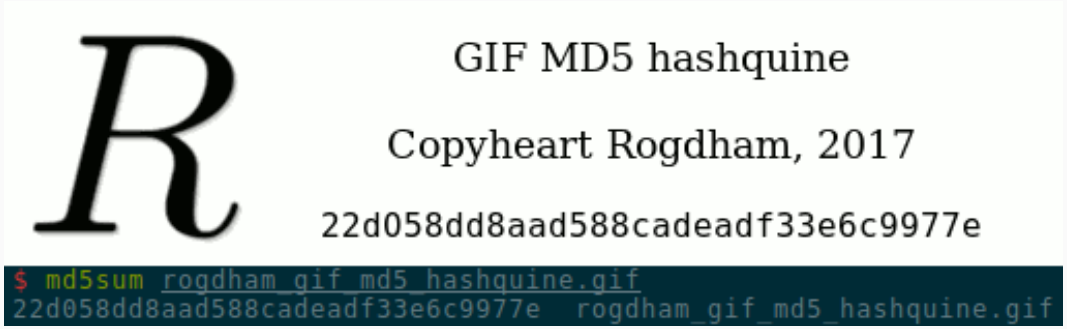
**Theorem**
If the compression function $f$ is collision resistant, and if the padding is done correctly, then the obtained hash function $h$ is collision resistant.

# Merkle-Damgård weaknesses

- Second preimage attacks are more efficient than bruteforce
- Multicollisions (several texts that have the same hash) are not much harder than simple collisions
- Length extension attacks

# MD5 and SHA weaknesses

MD5 is quite easy to attack. Attacks found in 1996. First collision in 2004. Nowadays, finding a collision takes less than a minute on a laptop. Some people even have fun:




SHA-1 is getting there too. First attack discovered in 2005, first collision in 2017.

# Call for a new hash

In 2007, NIST calls for a new family of hash functions (what would become SHA-3).

They should have original architecture, be flexible in the design.

51 participants, some were proven weak quickly, some where eliminated for complicated design etc.

The winner is Keccak, in 2012. It relies on the novel (2006-2007) sponge construction.

# Sponge Construction

Let $f$ be a permutation fonction or a transformation

$$f : \{0,1\}^n \rightarrow \{0,1\}^n$$

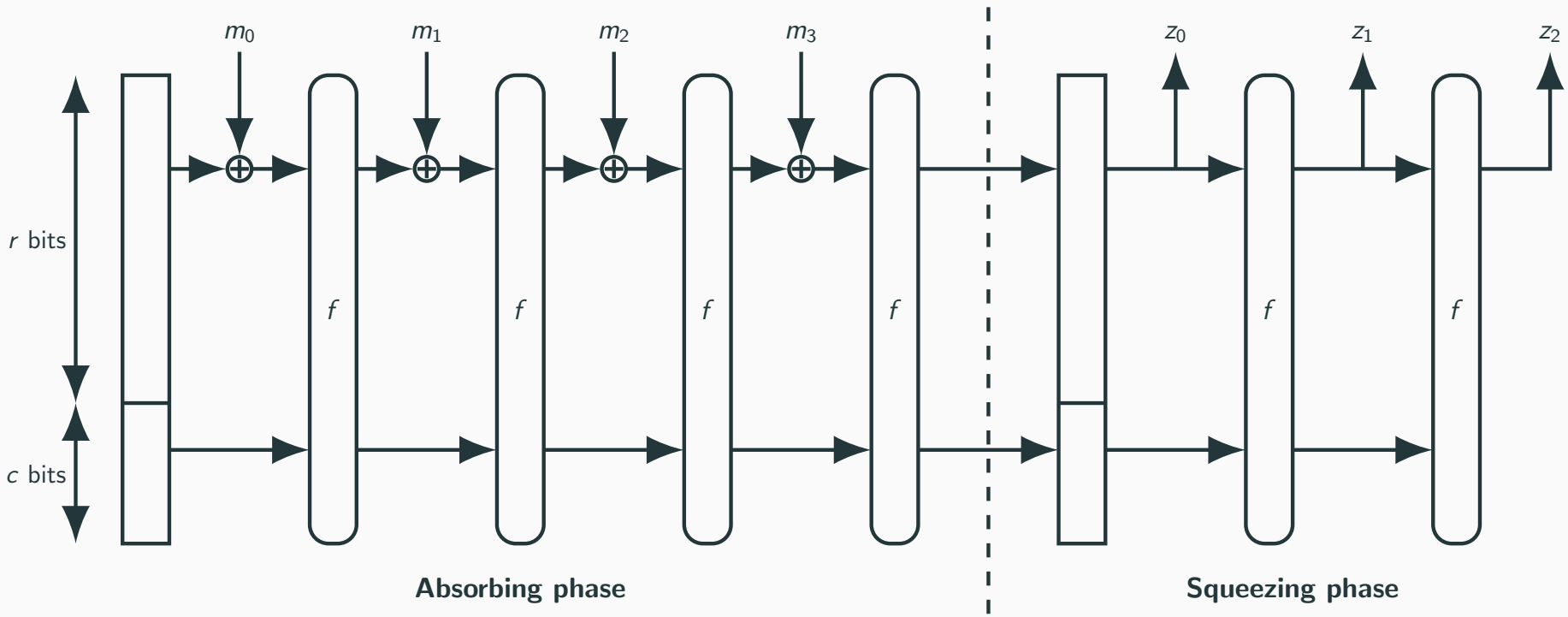
1. Break the message $m$ to hash in blocks of size $r$ ($r < n$)

$$m = m_0 m_1 \dots m_t$$

2. Pad $m_t$ with zeros as necessary
3. Iterate over the blocks

$$\begin{aligned} r_0 &= 0^r \\ c_0 &= 0^{n-r} \\ c_i \| r_i &= f(c_{i-1} || (r_{i-1} \oplus m_{i-1})) \\ h(m) &= c_t \| r_t \end{aligned}$$

# Sponge Construction




## Example

- SHA-3

# Use of Cryptographic Hash Functions

Compute a digest $y$ from a given message $m$. The digest should be specific to the message.

**Examples**

- Use $y$ in place of $m$ in a trustworthy way.
- *"Did you get m correctly? Here's y to check."* (file-sharing)
- *"I sign y to prove that I wrote m."*

# MAC

# MAC (Message Authentication Code)

**Idea**

A MAC guarantees the integrity of the message and authenticates the sender via a secret key. The MAC is sent along with the message. The message is sent in cleartext (not encrypted).

**Difference with Hash Functions**

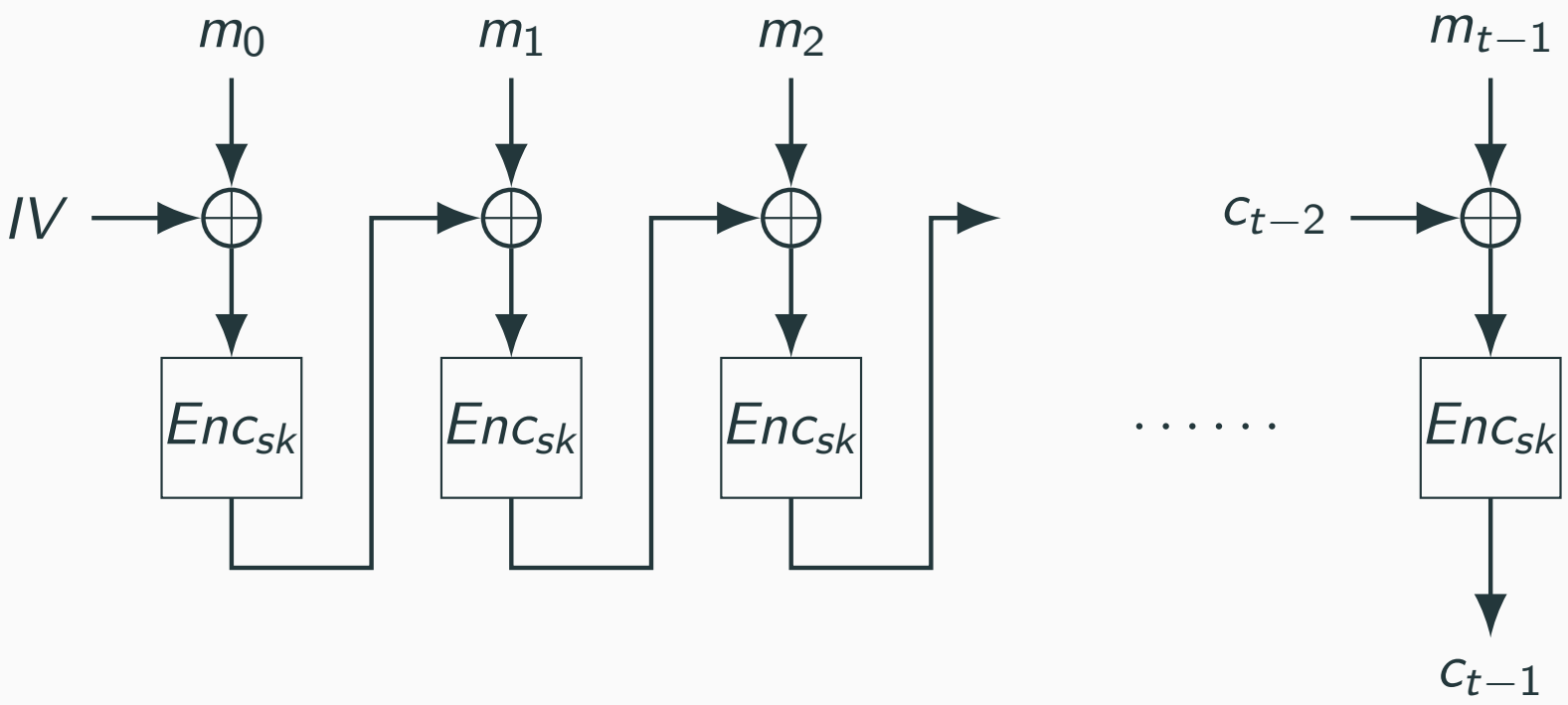
We need a *secret key* to verify the integrity of the message. This guarantees the authenticity of the message.

# MAC (Message Authentication Code)


$$m$$
$$\mathrm{MAC}_{sk}(m) = \mathrm{mac}$$

$$\mathrm{mac} \stackrel{?}{=} \mathrm{MAC}_{sk}(m)$$

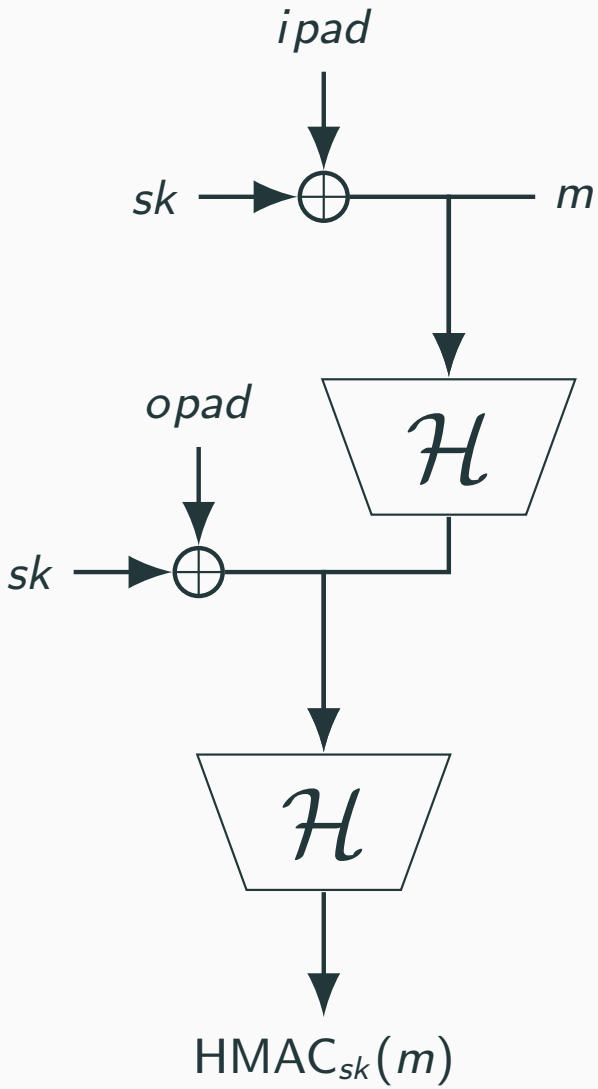
# MAC Based on Block Cipher (CBC-MAC)




Then use bits 16 to 64 of $c_{t-1}$ as CBC-MAC (according to NIST standards)

Not secure with variable-length messages.

# MAC Based on Hash Function (HMAC)




With $ipad = 0x36363636\ldots36$ and $opad = 0x5c5c5c5c\ldots5c$. Formalized in RFC2014, used in IPSec, TLS...

# Authenticated encryption

We want both confidentiality, integrity and authenticity (this is called **authenticated encryption**).

Which one to chose?

- Encrypt-then-MAC
- MAC-then-encrypt (SSL/TLS)
- Encrypt-and-MAC (SSH)

# Authenticated encryption

We want both confidentiality, integrity and authenticity (this is called **authenticated encryption**).

Which one to chose?

- Encrypt-then-MAC
- MAC-then-encrypt (SSL/TLS)
- Encrypt-and-MAC (SSH)

According to Bellare and Namprempre (2000), Encrypt-then-MAC has best security guarantees.

# Authenticated encryption modes

Suppose we have a message $m$, a MAC function $MAC$, a secret key sk and a symmetric cipher, with encryption $Enc$.

- Encrypt-then-MAC: send $Enc_{\mathsf{sk}}(m), MAC_{\mathsf{sk}}(Enc_{\mathsf{sk}}(m))$
- MAC-then-encrypt: send $Enc_{\mathsf{sk}}(m, MAC_{\mathsf{sk}}(m))$
- Encrypt-and-MAC: send $Enc_{\mathsf{sk}}(m), MAC_{\mathsf{sk}}(m)$

# Things to remember

- Properties of a cryptographic hash function
- MD5, SHA-1, Keccak
- Basic concept of Merkle-Damgård, sponge construction
- MAC, HMAC
- Authenticated encryption, Encrypt-then-MAC

# Thank you for your attention

**Any question?**